PreProcessing On dirty_iris Dataset

October 6, 2025

[1]:
```
from IPython.display import display
import ipywidgets as widgets

uploader = widgets.FileUpload(accept='', multiple=False)
display(uploader)
```

```
FileUpload(value=(), description='Upload')
```

[3]:
```
# Get the uploaded file from the tuple
uploaded_file = uploader.value[0]

# Read the CSV data into a pandas DataFrame
import pandas as pd
import io

df = pd.read_csv(io.BytesIO(uploaded_file['content']))

# Display the first five rows to check
df.head()
```

[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	NaN	0.2	setosa
2	4.7	3.2	NaN	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[4]:
```
# Calculate the number and percentage of complete (non-missing) observations

# Count rows with no missing values
num_complete = df.dropna().shape[0]
```

```
# Count total rows
num_total = df.shape[0]

# Calculate percentage of complete rows
percent_complete = (num_complete / num_total) * 100

print(f"Number of complete observations: {num_complete}")
print(f"Percentage of complete observations: {percent_complete:.2f}%")
```

```
Number of complete observations: 88
Percentage of complete observations: 58.67%
```

[6]:
```
# Count the total number of rows (observations)
total_obs = df.shape[0]

# Count the number of complete rows (no missing data)
complete_obs = df.dropna().shape[0]

# Count rows with at least one missing value
incomplete_obs = df.isnull().any(axis=1).sum()

print(f"Total observations: {total_obs}")
print(f"Complete observations: {complete_obs}")
print(f"Incomplete observations: {incomplete_obs}")
```

```
Total observations: 150
Complete observations: 88
Incomplete observations: 62
```

[7]:
```
# Replace special values (like '?', '-', 'N/A', 'null') with NaN (missing value)
import numpy as np

# List of special values to consider as missing
special_values = ['?', '-', 'N/A', 'null', 'NA']

# Replace in the entire DataFrame
df.replace(special_values, np.nan, inplace=True)

print("Special values replaced with NaN (missing value).")
```

```
Special values replaced with NaN (missing value).
```

[8]:
```
# Display rows that have at least one missing value
print(df[df.isnull().any(axis=1)].head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
1                4.9               3.0                NaN               0.2
2                4.7               3.2                NaN               0.2
```

```
7                5.0               NaN               1.5               0.2
8                NaN               2.9               1.4               0.2
9                4.9               NaN               1.5               0.1
```

```
  species
1  setosa
2  setosa
7  setosa
8  setosa
9  setosa
```

[9]:
```
# Print the number of missing (NaN) values in each column
print(df.isnull().sum())
```

```
sepal length (cm)    23
sepal width (cm)     15
petal length (cm)    18
petal width (cm)     16
species               0
dtype: int64
```

[10]:
```
# Create rules.txt file with the validation rules
rules_text = """Species: setosa, versicolor, virginica
All numeric properties positive
Petal Length >= 2 x Petal Width
Sepal Length <= 30
Sepal Length > Petal Length"""

with open('rules.txt', 'w') as f:
    f.write(rules_text)
print("Rules file created.")
```

```
Rules file created.
```

[11]:
```
# Read rules from text file and print them to verify
with open('rules.txt', 'r') as f:
    rules_loaded = f.read()

print("Loaded validation rules:")
print(rules_loaded)
```

```
Loaded validation rules:
Species: setosa, versicolor, virginica
All numeric properties positive
Petal Length >= 2 x Petal Width
Sepal Length <= 30
Sepal Length > Petal Length
```

[12]:
```
# Define Python functions for each rule, to later check violations
def rule_species(row):
    return row['species'] in ['setosa', 'versicolor', 'virginica']

def rule_positive(row):
    return all(float(row[col]) > 0 for col in ['sepal length (cm)', 'sepal␣
 ↪width (cm)', 'petal length (cm)', 'petal width (cm)'])

def rule_petal_length(row):
    return float(row['petal length (cm)']) >= 2 * float(row['petal width (cm)'])

def rule_sepal_length(row):
    return float(row['sepal length (cm)']) <= 30

def rule_sepal_vs_petal(row):
    return float(row['sepal length (cm)']) > float(row['petal length (cm)'])

# Collect rules in a constraint object (here a list for simplicity)
constraints = [rule_species, rule_positive, rule_petal_length,␣
 ↪rule_sepal_length, rule_sepal_vs_petal]

print("Constraint object created!")
print(constraints)
```

```
Constraint object created!
[<function rule_species at 0x0000012BE32C96C0>, <function rule_positive at
0x0000012BE345FEC0>, <function rule_petal_length at 0x0000012BE42287C0>,
<function rule_sepal_length at 0x0000012BE4228680>, <function
rule_sepal_vs_petal at 0x0000012BE4228540>]
```

[13]:
```
# Print the resulting constraint object (list of rule functions)
print("Constraint object (list of rule functions):")
for idx, rule in enumerate(constraints, 1):
    print(f"Rule {idx}: {rule.__name__}")
```

```
Constraint object (list of rule functions):
Rule 1: rule_species
Rule 2: rule_positive
Rule 3: rule_petal_length
Rule 4: rule_sepal_length
Rule 5: rule_sepal_vs_petal
```

[14]:
```
# Count how often each rule is broken
violations = {f"Rule {i+1}": 0 for i in range(len(constraints))}

# Loop through each row in the dataset
for _, row in df.iterrows():
    for i, rule in enumerate(constraints):
```

```
        try:
            if not rule(row):
                violations[f"Rule {i+1}"] += 1
        except:
            violations[f"Rule {i+1}"] += 1  # treat error (e.g. missing values)␣
 ↪as a violation

# Print how many times each rule was broken
print("Rule violations summary:")
for rule, count in violations.items():
    print(f"{rule} broken: {count} times")
```

```
Rule violations summary:
Rule 1 broken: 0 times
Rule 2 broken: 62 times
Rule 3 broken: 31 times
Rule 4 broken: 23 times
Rule 5 broken: 37 times
```

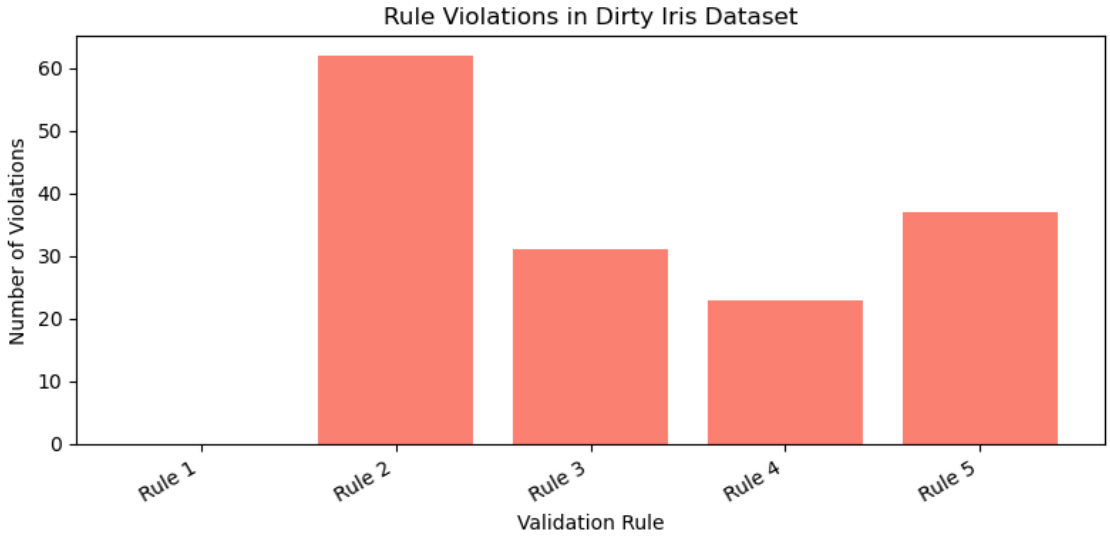
[15]:
```
# You must use the actual column names in your CSV, e.g.:
# 'sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width␣
 ↪(cm)', 'species'

def rule_species(row):
    return row['species'] in ['setosa', 'versicolor', 'virginica']

def rule_positive(row):
    try:
        return all(float(row[col]) > 0 for col in [
            'sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',␣
 ↪'petal width (cm)'
        ])
    except:
        return False

def rule_petal_length(row):
    try:
        return float(row['petal length (cm)']) >= 2 * float(row['petal width␣
 ↪(cm)'])
    except:
        return False

def rule_sepal_length(row):
    try:
        return float(row['sepal length (cm)']) <= 30
    except:
        return False
```

```
def rule_sepal_vs_petal(row):
    try:
        return float(row['sepal length (cm)']) > float(row['petal length (cm)'])
    except:
        return False

constraints = [rule_species, rule_positive, rule_petal_length,␣
 ↪rule_sepal_length, rule_sepal_vs_petal]

violations = {f"Rule {i+1}": 0 for i in range(len(constraints))}
for _, row in df.iterrows():
    for i, rule in enumerate(constraints):
        if not rule(row):
            violations[f"Rule {i+1}"] += 1

print("Rule violations summary:")
for rule, count in violations.items():
    print(f"{rule} broken: {count} times")
```

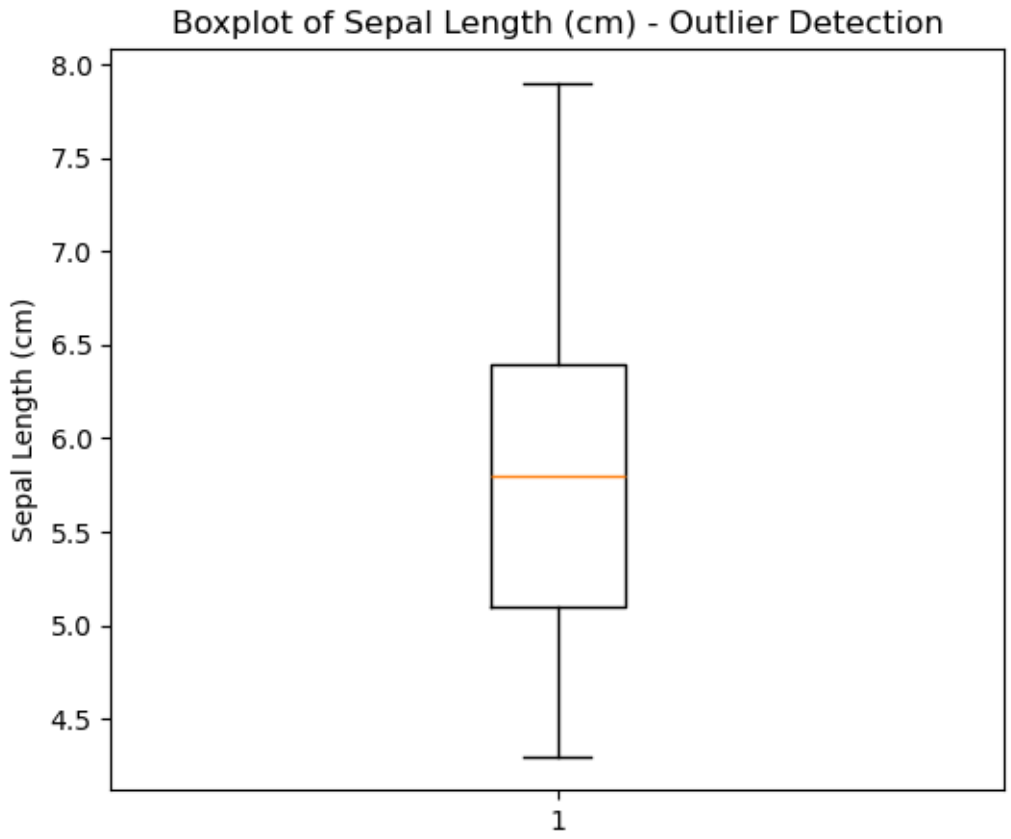
```
Rule violations summary:
Rule 1 broken: 0 times
Rule 2 broken: 62 times
Rule 3 broken: 31 times
Rule 4 broken: 23 times
Rule 5 broken: 37 times
```

[19]:
```
import matplotlib.pyplot as plt  # Import the plotting library

# Convert your violations dictionary keys and values to separate lists for␣
 ↪plotting
plot_labels = list(violations.keys())      # X-axis: rule descriptions
plot_values = list(violations.values())    # Y-axis: violation counts

# Set up the size of the plot
plt.figure(figsize=(8,4))

# Create a bar plot with the rule labels and number of violations
plt.bar(plot_labels, plot_values, color='salmon')

# Label the axes and the chart
plt.ylabel("Number of Violations")          # Label for y-axis
plt.xlabel("Validation Rule")               # Label for x-axis
plt.title("Rule Violations in Dirty Iris Dataset")  # Chart title

# Rotate the x-axis labels for better readability
plt.xticks(rotation=30, ha='right')
```

```
plt.tight_layout()  # Ensure everything fits without overlap
plt.show()          # Display the plot
```



[20]:
```
# v) Find outliers in sepal length using boxplot

import matplotlib.pyplot as plt  # Import plotting library

# Plot a boxplot for the 'sepal length (cm)' column
plt.figure(figsize=(6, 5))
plt.boxplot(df['sepal length (cm)'].dropna())  # Drop NaN values to avoid errors

plt.ylabel("Sepal Length (cm)")  # Y-axis label
plt.title("Boxplot of Sepal Length (cm) - Outlier Detection")  # Title
plt.show()
```



[ ]:
```
!jupyter nbconvert --to pdf "PreProcessing On dirty_iris Dataset.ipynb"␣
↪--output "C:/Users/ASUS/Downloads/preprocessing_on_dirty_iris_dataset.pdf"
```

[ ]:
```

```